X.509 InhibitPolicyMapping Processing

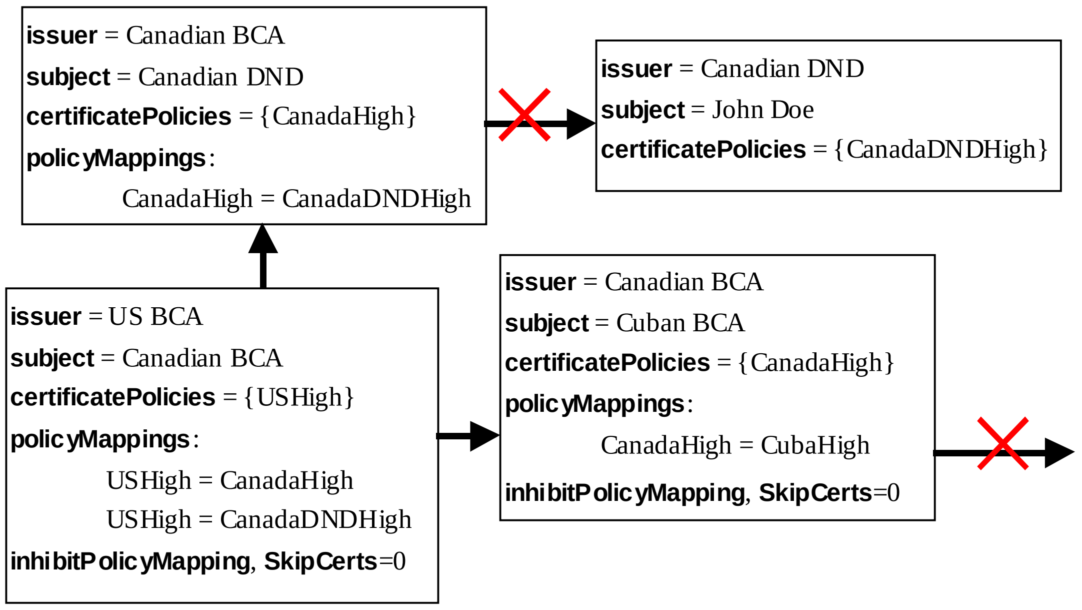
The Problem

Currently, when the **inhibitPolicyMapping** extension is included in a certificate or the *initial-policy-mapping-indicator* is set by the relying party, all policy mapping is turned off. The result of the current certificate policy processing rules (with respect to the **inhibitPolicyMapping** extension) is to cause some paths to be rejected even though some relying parties would like to validate such paths. This problem occurs whether using the current certificate policy processing rules, the rules suggested by Santosh Chokhani, Serge Mister, and Tim Moses (see http://csrc.nist.gov/pki/twg/presentations/twg-99-31.pdf), or the rules suggested by Sarbari Gupta (see http://csrc.nist.gov/pki/twg/presentations/twg-99-32.pdf).

In order to illustrate the problem with the processing of the **inhibitPolicyMapping** extension, we will use the following example. The US BCA wishes to cross-certify with the Canadian BCA. Relying parties within the US wish to be able to validate certification paths to Canadian certificate subjects, even if they happen to have been issued a certificate under a different policy than the one used by the Canadian BCA. At the same time, paths from the Canadian BCA to non-Canadian certificate subjects should not validate for US relying parties.

In the figure below, the Canadian BCA has issued two certificates, one to the Canadian DND and one to the Cuban BCA. The Canadian BCA issues each certificate under the policy CanadaHigh and then uses policy mapping to map CanadaHigh to the appropriate policy for the domain of the subject of the certificate. In each case the Canadian BCA relies on policy mapping to allow Canadian relying parties to validate paths within Cuba and the Canadian DND.

The figure below also includes a certificate issued by the US BCA to the Canadian BCA. In order to prevent US relying parties from validating paths that lead outside of Canada, the US BCA turns off policy mapping.



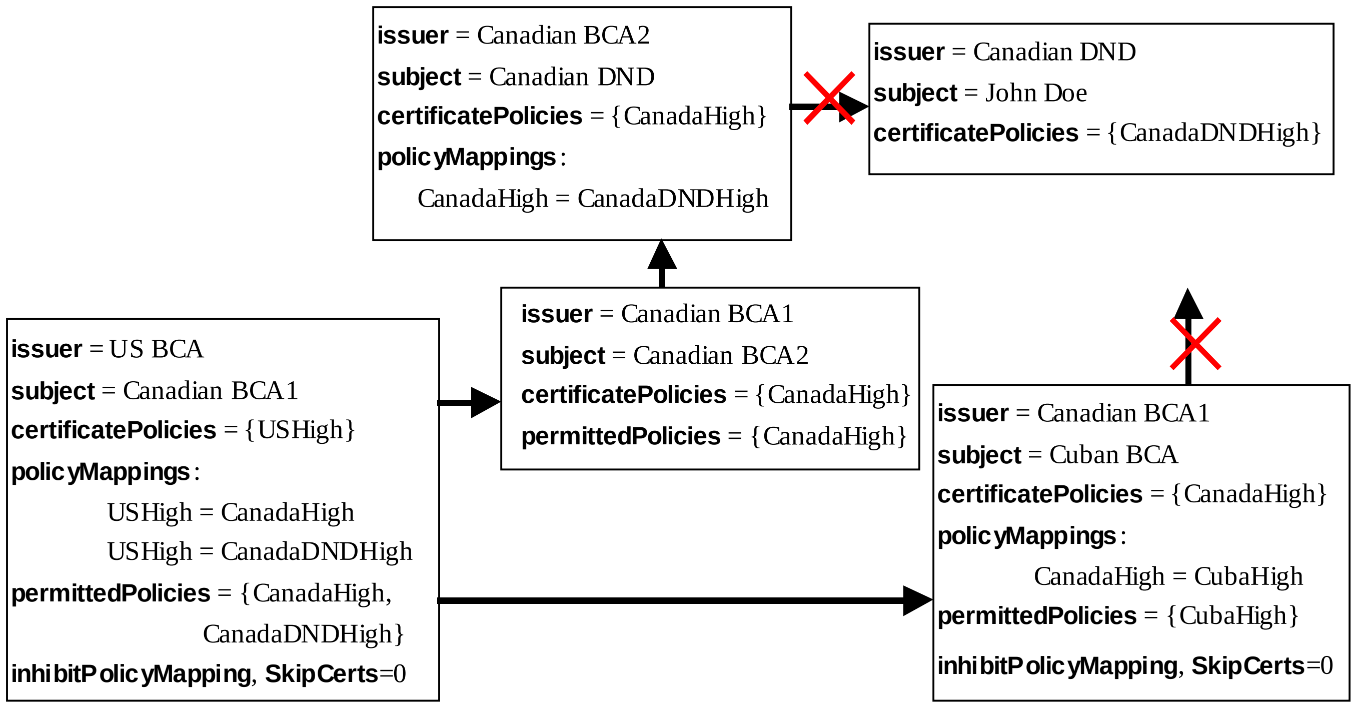
issuer = Canadian BCA
subject = Canadian DND
certificatePolicies = {CanadaHigh}
policyMappings:
CanadaHigh = CanadaDNDHigh
issuer = Canadian DND
subject = John Doe
certificatePolicies = {CanadaDNDHigh}
issuer = US BCA
subject = Canadian BCA
certificatePolicies = {USHigh}
policyMappings:
USHigh = CanadaHigh
USHigh = CanadaDNDHigh
inhibitPolicyMapping, **SkipCerts**=0
issuer = Canadian BCA
subject = Cuban BCA
certificatePolicies = {CanadaHigh}
policyMappings:
CanadaHigh = CubaHigh
inhibitPolicyMapping, **SkipCerts**=0

We will now describe the results of certificate path processing on the certificate path beginning with the US BCA and ending with the certificate issued to John Doe when either the current rules are used or when the rules suggested by Santosh Chokhani, Serge Mister, and Tim Moses are used. In both cases, we will assume that all certificate policy extensions are critical and that *initial-policy-set* is *any-policy*.

Current rules: Under the current rules, *authority-constrained-policy-set* will initially be *any-policy*. After processing the US BCA certificate, *authority-constrained-policy-set* will become {USHigh, CanadaHigh, CanadaDNDHigh}. When processing the Canadian BCA certificate, *authority-constrained-policy-set* will be set to the intersection of its current value and the value in **certificatePolicies**, {CanadaHigh}. Since the US BCA included the **inhibitPolicyMapping** extension, the value of *authority-constrained-policy-set* will still be {CanadaHigh} when processing of the Canadian BCA certificate is complete. When processing the certificate issued to John Doe, the intersection of *authority-constrained-policy-set* and **certificatePolicies** will be the empty set and so the certificate will be rejected.

New Rules: Under the rules proposed by Santosh Chokhani, Serge Mister, and Tim Moses, *authority-constrained-policy-set* will initially be *any-policy*. After processing the US BCA certificate, *authority-constrained-policy-set* will become {CanadaHigh, CanadaDNDHigh}. When processing the Canadian BCA certificate, *authority-constrained-policy-set* will be set to the intersection of its current value and the value in **certificatePolicies**, {CanadaHigh}. Since the US BCA included the **inhibitPolicyMapping** extension, the processing of the **policyMapping** extension will result in CanadaHigh being deleted from *authority-constrained-policy-set* with the result being that *authority-constrained-policy-set* becomes the empty set. After processing the certificate issued to John Doe, the certificate path will be rejected since *authority-constrained-policy-set* is empty.

A similar problem occurs when using the certificate policy processing rules proposed by Sarbari Gupta, however, in order to demonstrate this a slightly different set of certificates will need to be used. In the figure below, the US BCA has issued a certificate to Canadian BCA1, which issues a certificate under the policy CanadaHigh to Canadian BCA2. By using the **permittedPolicies** extension, Canadian BCA1 restricts the certificates that can be validated to those that assert CanadaHigh and to those that assert a policy that is at some point mapped to CanadaHigh using the **policyMappings** extension. Since Canadian BCA2 asserts that CanadaDNDHigh is equivalent to CanadaHigh, Canadian relying parties, whose certificate paths include Canadian BCA1 and Canadian BCA2 can validate the certificate issued to John Doe.


issuer = Canadian BCA2
subject = Canadian DND
certificatePolicies = {CanadaHigh}
policyMappings:
CanadaHigh = CanadaDNDHigh
issuer = Canadian DND
subject = John Doe
certificatePolicies = {CanadaDNDHigh}
issuer = Canadian BCA1
subject = Canadian BCA2
certificatePolicies = {CanadaHigh}
permittedPolicies = {CanadaHigh}
issuer = US BCA
subject = Canadian BCA1
certificatePolicies = {USHigh}
policyMappings:
USHigh = CanadaHigh
USHigh = CanadaDNDHigh
permittedPolicies = {CanadaHigh, CanadaDNDHigh}
inhibitPolicyMapping, SkipCerts=0
issuer = Canadian BCA1
subject = Cuban BCA
certificatePolicies = {CanadaHigh}
policyMappings:
CanadaHigh = CubaHigh
permittedPolicies = {CubaHigh}
inhibitPolicyMapping, SkipCerts=0

A different result occurs when a US relying party attempts to validate the certificate issued to John Doe. As above, we will assume that *initial-policy-set* is set to *any-policy*. After processing the US BCA certificate, *authority-constrained-policy-set* will be {CanadaHigh, CanadaDNDHigh}. When processing the Canadian BCA1 certificate, *authority-constrained-policy-set* will be set to the intersection of its current value and the contents of the **permittedPolicies** extension, {CanadaHigh}. Since the US BCA included the **inhibitPolicyMapping** extension and Canadian BCA2 did not include a **permittedPolicies** extension, *authority-constrained-policy-set* will remain unchanged by the processing of the Canadian BCA2 certificate. When processing the certificate issued to John Doe, it will be determined that no member of *authority-constrained-policy-set* appears in the **certificatePolicies** extension and so the certificate path will be rejected.

A Solution

The problem in each of these cases was that the Canadian CAs issued certificates to other Canadian CAs under the assumption that policy mapping was enabled. In reality, however, policy mapping was disabled for US relying parties. The result is that US relying parties are unable to validate some Canadian certificates even though the US BCA asserts that all Canadian certificate policies are permitted.

One solution to the policy mapping problem is to use the **inhibitPolicyMapping** extension to turn policy mapping off on a CA-by-CA basis. Following this solution, there would be three *constrained-policy* sets for the example above (four if the relying party's *initial-policy-set* is taken into account).

In the example, the value of *US-BCA-constrained-policy-set* would initially be set to {CanadaHigh, CanadaDNDHigh}. When processing the certificate issued by the Canadian BCA1 to the Canadian BCA2, *Canadian-BCA1-constrained-policy-set* would be set to {CanadaHigh}. When processing the certificate issued by Canadian BCA2 to Canadian DND, *Canadian-BCA2-constrained-policy-set* would be set to *any-policy*.[1] Since the US BCA included the **inhibitPolicyMapping** extension in its certificate, *US-BCA-constrained-policy-set* would remain unchanged. However, CanadaDNDHigh would be added to *Canadian-BCA1-constrained-policy-set* since the Canadian BCA1 did not include the **inhibitPolicyMapping** extension in its certificate. When processing the certificate issued to John Doe, the intersection of *US-BCA-constrained-policy-set*, *Canadian-BCA1-constrained-policy-set*, and *Canadian-BCA2-constrained-policy-set* would be {CanadaHigh, CanadaDNDHigh}. Since the certificate issued to John Doe asserts one of these policies in its **certificatePolicies** extension, the path will validate.

When processing the certificate issued to the Cuban BCA, the results would still be as desired. After processing both the certificate issued by the US BCA and the certificate issued by the Canadian BCA1 to the Cuban BCA, *US-BCA-constrained-policy-set* would be {CanadaHigh, CanadaDNDHigh} and *Canadian-BCA1-constrained-policy-set* would be {CubaHigh}. Since the intersection of *US-BCA-constrained-policy-set* and *Canadian-BCA1-constrained-policy-set* is the empty set, any certificates issued by Cuba would be rejected by US relying parties.

A proposed set of certificate path processing rules that take into account the **permittedPolicies** extension as well as the proposal for handling the **inhibitPolicyMapping** extension as described above is included below.

Conclusion

The current processing rules may limit the types of certificate paths that can be processed. An examination of business needs will be necessary to determine whether this is a feature or a bug.

[1] This is only one option for setting a CAs *constrained-policy-set* value when the **permittedPolicies** extension is not present. As is shown below other options are possible, but choosing a different value for *Canadian-BCA2-constrained-policy-set*, as long as it contained CanadaDNDHigh, would not affect the outcome of certificate policy processing in this example.

12.4.2.3 Permitted Policies field

This field, which shall be used only in a CA-certificate, indicates the set of certificate policies under which all subsequent certificates must be issued. This field is defined as follows:

```
permittedPolicies EXTENSION ::= {
        SYNTAX          permittedPoliciesSyntax
        IDENTIFIED BY id-ce-permittedPolicies }

permittedPoliciesSyntax ::= SEQUENCE SIZE (1..MAX) of PolicyInformation
```

If this extension is present then all subsequent certificates that include a **certificatePolicies** extension must include in that extension at least one of the policies specified in this extension.

This extension is always critical.

12.4.3 Certification path processing procedure

Certification path processing is carried out in a system which needs to use the public key of a remote end entity, e.g., a system which is verifying a digital signature generated by a remote entity. The certificate policies, basic constraints, name constraints, and policy constraints extensions have been designed to facilitate automated, self-contained implementation of certification path processing logic.
Following is an outline of a procedure for validating certification paths. An implementation shall be functionally equivalent to the external behavior resulting from this procedure. The algorithm used by a particular implementation to derive the correct output(s) from the given inputs is not standardized.

The inputs to the certification path processing procedure are:

a) a set of certificates comprising a certification path;

b) a trusted public key value or key identifier (if the key is stored internally to the certification path processing module), for use in verifying the first certificate in the certification path;

c) an *initial-policy-set* comprising one or more certificate policy identifiers, indicating that any one of these policies would be acceptable to the certificate user for the purposes of certification path processing; this input can also take the special value *any-policy*;

d) an *initial-explicit-policy* indicator value, which indicates if an acceptable policy identifier needs to explicitly appear in the certificate policies extension field of all certificates in the path;

e) an *initial-policy-mapping-inhibit* indicator value, which indicates if policy mapping is forbidden in the certification path; and

f) the current date/time (if not available internally to the certification path processing module).

The values of c), d), and e) will depend upon the policy requirements of the user-application combination that needs to use the certified end-entity public key.

The outputs of the procedure are:

a) an indication of success or failure of certification path validation;

b) if validation failed, a diagnostic code indicating the reason for failure;

c) if validation was successful, a set of policies constrained by the CAs in the certification path under which each certificate in the path may be used, together with all qualifiers for these policies, or the special value *any-policy*. Unless *any-policy* is returned, the certificate user shall only use the certificate in accordance with one of the identified policies and shall process all qualifiers for that policy present in the certification path.

d) if validation was successful and c) returned the value *any-policy*, the set of all policy element qualifiers encountered in the certification path.

e) details of any policy mapping that occurred in processing the certification path.

NOTE — If validation is successful, the certificate-using system may still choose not to use the certificate as a result of values of policy qualifiers or other information in the certificate.

The procedure makes use of the following set of state variables:

a) *acceptable-policies*[*i*]*:* The set of certificate policies under which the i-th certificate in the certification path may be used;

b) *constrained-policy-set*[*i*]*:* A set of certificate policy identifiers comprising the set of policies currently considered acceptable by the *i*-th certificate in the certification path; this state variable can also take the special value *any-policy*;

c) *permitted-subtrees:* A set of subtree specifications defining subtrees within which all subject names in subsequent certificates in the certification path must fall, or may take the special value *unbounded*;

d) *excluded-subtrees:* A (possibly empty) set of subtree specifications (each comprising a subtree base name and maximum and minimum level indicators) defining subtrees within which no subject name in a subsequent certificate in the certification path may fall;

e) *explicit-policy-indicator:* Indicates if an acceptable policy needs to be explicitly identified in every certificate;

f) *policy-mapping-inhibit-indicator*[*i*]*:* Indicates if policy mapping is inhibited by the i-th certificate in the certificate path;

g) *explicit-policy-pending-constraints:* Details of explicit-policy constraints which have been stipulated but are yet to take effect. There is a one-bit indictor called *explicit-policy-pending* along with an integer called *explicit-policy-skip-certificates* which gives the number of certificates yet to skip before the constraint takes effect;

h) *policy-mapping-inhibit-pending-constraints*[*i*]*:* Details of inhibit-policy constraints which have been stipulated by the i-th certificate in the certificate path but are yet to take effect. There is a one-bit indicator called *policy-mapping-inhibit-pending*[*i*] along with an integer *called policy-mapping-inhibit-skip-certificates*[*i*] which gives the number of certificates yet to skip before the constraint takes effect;

i) *path-depth:* An integer equal to the number of certificates in the certification path for which processing has been completed.

The procedure involves an initialization step, followed by a series of certificate-processing steps. The initialization step comprises:

a) Initialize *constrained-policy-set*[*0*] to the value of *initial-policy-set*;

b) Initialize the *permitted-subtrees* variable to *unbounded*;

d) Initialize the *excluded-subtrees* variable to an empty set;

e) Initialize the *explicit-policy-indicator* to the *initial-explicit-policy* value;

f) Initialize *policy-mapping-inhibit-indicator*[*0*] to the *initial-policy-mapping-inhibit* value;

g) Initialize the *explicit-policy-pending* and *policy-mapping-inhibit-pending*[*0*] indicators to unset;

h) Initialize *path-depth* to 0.

Each certificate is then processed in turn, starting with the certificate signed using the input trusted public key. The last certificate is considered to be the *end certificate*; any other certificates are considered to be *intermediate certificates.*

The following checks are applied to a certificate:

a) Check that the signature verifies, that dates are valid, that the certificate subject and certificate issuer names chain correctly, and that the certificate has not been revoked.

b) For an intermediate certificate, if the **basicConstraints** extension is present in the certificate, check that the **cA** component is present and set to true. If the **pathLenConstraint** component is present, check that the current certification path does not violate that constraint.

c) If *explicit-policy-indicator* is set, check that the **certificatePolicies** extension is present.

d) Compute the intersection of *constrained-policy-set*[*0*] … *constrained-policy-set*[*path-depth*] and put the result as the value of *acceptable-policies*[*path-depth*+1]. Check that *acceptable-policies*[*path-depth*+1] is non-empty.

e) If the **certificatePolicies** extension is present and the value in the extension is not **anyPolicy**, compute the intersection of the policies in that extension and *acceptable-policies*[*path-depth*+1] and put the result as the new value of *acceptable-policies*[*path-depth*+1]. Check that *acceptable-policies*[*path-depth*+1] is non-empty.

f) For each *i* between 0 and *path-depth*:
For each certificate policy in *constrained-policy-set*[*i*]:

— If the certificate policy is not **anyPolicy**, add any policy qualifiers attached to the policy to the corresponding certificate policy in *acceptable-policies*[*path-depth*+1].

— If the certificate policy is **anyPolicy**, add any policy qualifiers attached to the policy to each certificate policy in *acceptable-policies*[*path-depth*+1].

g) If the **certificatePolicies** extension is present and the value in the extension is not **anyPolicy**, add any policy qualifiers from the **certificatePolicies** extension to the corresponding certificate policy in *acceptable-policies*[*path-depth*+1].

h) If the **certificatePolicies** extension is present and the value in the extension is **anyPolicy**, add any policy qualifiers from the **certificatePolicies** extension to each certificate policy in *acceptable-policies*[*path-depth*+1].

i) Check that the subject name is within the name-space given by the value of *permitted-subtrees* and is not within the name-space given by the value of *excluded-subtrees*.

If any of the above checks fails, the procedure terminates, returning a failure indication and an appropriate reason code. If none of the above checks fails on the end certificate, the procedure terminates, returning a success indication together with the set of policy identifiers from *acceptable-policies*, the required policy element qualifiers, and details of any policy mapping that may have occurred.

For an intermediate certificate, the following constraint recording actions are then performed, in order to correctly set up the state variables for the processing of the next certificate

a) If the **nameConstraints** extension with a **permittedSubtrees** component is present in the certificate, set the *permitted-subtrees* state variable to the intersection of its previous value and the value indicated in the certificate extension.

b) If the **nameConstraints** extension with an **excludedSubtrees** component is present in the certificate, set the *excluded-subtrees* state variable to the union of its previous value and the value indicated in the certificate extension.

c) If *explicit-policy-indicator* is not set:

— if the *explicit-policy-pending* indicator is set, decrement *explicit-policy-skip-certificates* and, if this value becomes zero, set *explicit-policy-indicator*.

— If the **requireExplicitPolicy** constraint is present in the certificate perform the following. For a **SkipCerts** value of 0, set *explicit-policy-indicator*. For any other **SkipCerts** value, set the *explicit-policy-pending* indicator, and set *explicit-policy-skip-certificates* to the lesser of the **SkipCerts** value and the previous *explicit-policy-skip-certificates* value (if the *explicit-policy-pending* indicator was already set).

d) For each *i* between 0 and *path-depth*:

If *policy-mapping-inhibit-indicator*[*i*] is not set:

— process any policy mapping extension with respect to policies in the *constrained-policy-set*[*i*] and add the appropriate policy identifiers to *constrained-policy-set*[*i*].

— if the *policy-mapping-inhibit-pending*[*i*] indicator is set, decrement *policy-mapping-inhibit-skip-certificates*[*i*] and, if this value becomes zero, set the *policy-mapping-inhibit-indicator*[*i*].

e) Increment *path-depth*.

f) If the **permittedPolicies** extension is present in the certificate, set *constrained-policy-set*[*path-depth*] to the policies in that extension. Add any policy qualifiers from the **permittedPolicies** extension to the corresponding certificate policy in *constrained-policy-set*[*path-depth*].

g) If the **permittedPolicies** extension is not present in the certificate, [see **note on Step g** below].

h) If the **inhibitPolicyMapping** constraint is present in the certificate, perform the following. For a **SkipCerts** value of 0, set *policy-mapping-inhibit-indicator*[*path-depth*]. For any other **SkipCerts** value, set the *policy-mapping-inhibit-pending*[*path-depth*] indicator, and set *policy-mapping-inhibit-skip-certificates*[*path-depth*] to the value in **SkipCerts**.

Note on Step g:

Step g above specifies the value for *constrained-policy-set*[*path-depth*] when the **permittedPolicies** extension is not present in the certificate. Since a CA can set *constrained-policy-set*[*path-depth*] any way it wishes by using the **permittedPolicies** extension, step g can be anything. So, the only limitation in assigning a rule for step g is that it should be sensible. Below are three possibilities:

Option 1: If the **permittedPolicies** extension is not present in the certificate, set *constrained-policy-set*[*path-depth*] to *any-policy*.

Commentary: This is rule that Sarbari Gupta proposed in her presentation (see http://csrc.nist.gov/pki/twg/presentations/twg-99-32.pdf). This would be the preferred option if most CAs do not wish to limit the policies that can be used by CAs to which they issue certificates. It also seems to be the most intuitive and thus the option most likely to be understood by PKI users.

Option 2: If the **permittedPolicies** extension is not present but the **certificatePolicies** extension is present, set *constrained-policy-set*[*path-depth*] to the policies in the **certificatePolicies** extension. Add any policy qualifiers from the **certificatePolicies** extension to the corresponding certificate policy in *constrained-policy-set*[*path-depth*]. Process any policy mapping extension with respect to *constrained-policy-set*[*path-depth*] and add the appropriate policy identifiers to *constrained-policy-set*[*path-depth*].

If neither the **permittedPolicies** extension nor the **certificatePolicies** extension is present in the certificate, set *constrained-policy-set*[*path-depth*] to *any-policy*.

Commentary: Option 2 is the closest to the current version of the certificate policy processing rules, but given the current thinking on certificate policy processing, this may not be the best option.

Option 3: If the **permittedPolicies** extension is not present but the **certificatePolicies** extension is present, set *constrained-policy-set*[*path-depth*] to the policies in the **certificatePolicies** extension. Add any policy qualifiers from the **certificatePolicies** extension to the corresponding certificate policy in *constrained-policy-set*[*path-depth*]. Process any policy mapping extension with respect to *constrained-policy-set*[*path-depth*] by:

1. For each policy mapping in which the **issuerDomainPolicy** is in *constrained-policy-set*[*path-depth*] add the corresponding **subjectDomainPolicy** to *constrained-policy-set*[*path-depth*].
2. Remove from *constrained-policy-set*[*path-depth*] any policy which appears as an **issuerDomainPolicy** in one of the policy mappings.

If neither the **permittedPolicies** extension nor the **certificatePolicies** extension is present in the certificate, set *constrained-policy-set*[*path-depth*] to *any-policy*.

Commentary: The wording of this option needs work, but the idea is to be the same as option 2 with the exception that policy mapping causes that substitution of certificate policies instead of the accumulation of certificate policies. Option 3 is designed to match as closely as possible with the certificate policy processing proposal of Santosh Chokhani, Serge Mister, and Tim Moses (see http://csrc.nist.gov/pki/twg/presentations/twg-99-31.pdf). If most CAs wish to limit the set of acceptable policies to those listed in the **certificatePolicies** extension as modified by the **policyMapping** extension, then this may be the option that minimizes the number of certificates that will need to include the **permittedPolicies** extension.